# Homework #4

[ECE30021/ITP30002] Operating Systems

# Mission

- Solve problem 1, 2, and 3
  - Solve the problems on Ubuntu Linux (VirtualBox) using vim and gcc.
- Submission
  - Submit a .tgz file containing hw3_3_updated.c, hw4_2.c, and hw4_3.c on LMS.
    "tar cvfz hw4_<*student_id*>.tgz hw3_3_updated.c hw4_2.c hw4_3.c"
    (Do not copy&past but type the above command)
  - After compression, please check the .tgz file by decompressing in an empty directory.
    "tar xvfz hw4_<*student_id*>.tgz"
- Due date: PM 11:00:00, Apr. 25th

# Honor Code Guidelines

- “과제”
  - 과제는 교과과정의 내용을 소화하여 실질적인 활용 능력을 갖추기 위한 교육활동이다. 학생은 모든 과제를 정직하고 성실하게 수행함으로써 과제에 의도된 지식과 기술을 얻기 위해 최선을 다해야 한다.
  - 제출된 과제물은 성적 평가에 반영되므로 공식적으로 허용되지 않은 자료나 도움을 획득, 활용, 요구, 제공하는 것을 포함하여 평가의 공정성에 영향을 미치는 모든 형태의 부정행위는 단호히 거부해야 한다.
  - 수업 내용, 공지된 지식 및 정보, 또는 과제의 요구를 이해하기 위하여 동료의 도움을 받는 것은 부정행위에 포함되지 않는다. 그러나, 과제를 해결하기 위한 모든 과정은 반드시 스스로의 힘으로 수행해야 한다.
  - 담당교수가 명시적으로 허락한 경우를 제외하고 다른 사람이 작성하였거나 인터넷 등에서 획득한 과제물, 또는 프로그램 코드의 일부, 또는 전체를 이용하는 것은 부정행위에 해당한다.
  - 자신의 과제물을 타인에게 보여주거나 빌려주는 것은 공정한 평가를 방해하고, 해당 학생의 학업 성취를 저해하는 부정행위에 해당한다.
  - 팀 과제가 아닌 경우 두 명 이상이 함께 과제를 수행하여 이를 개별적으로 제출하는 것은 부정행위에 해당한다.
  - 스스로 많은 노력을 한 후에도 버그나 문제점을 파악하지 못하여 동료의 도움을 받는 경우도 단순한 문법적 오류에 그쳐야 한다. 과제가 요구하는 design, logic, algorithm의 작성에 있어서 담당교수, TA, tutor 이외에 다른 사람의 도움을 받는 것은 부정행위에 해당한다.
  - 서로 다른 학생이 제출한 제출물 간 유사도가 통상적으로 발생할 수 있는 정도를 크게 넘어서는 경우, 또는 자신이 제출한 과제물에 대하여 구체적인 설명을 하지 못하는 경우에는 부정행위로 의심받거나 판정될 수 있다.

# Problem 1

- Mission: peer review of HW#3 as a group
  - Sign up study group
    - Section 01: https://docs.google.com/spreadsheets/d/13c8OSrn7-l100vbSHuemXOCIPgH5tM9L3tftlvLdJ34/edit?usp=sharing
    - Section 02: https://docs.google.com/spreadsheets/d/1ozD9CYSYENmA7nza2jXrN4himsghODde25nUyffQDOE/edit?usp=sharing
  - Have an online meeting to review HW#3
    - Review the solutions of other members and share comments to each other.
    - Each member update solution of HW#3 problem 3 (hw3_3_updated.c) individually applying the comments.
      - Mark the modified parts by comments as the next page.
      - If you believe your previous solution was perfect, submit it again and put a comment “My previous solution was perfect, so I didn't modify it!” in the first line.

# Problem 1

- Example of updated solution.

```
int main()
{
	int i = 0;

	// previous code
	/*
	<old code>

	*/

	// updated code
	<new code>


	return 0;
}
```

# Problem 2

POSIX Shared Mem

- Search the Internet for the following structures and functions.
  - shm_open(), unlink()
  - ftruncate()
  - mmap(), munmap()

새로운 Shared Mem를 생성 or 기존의 것을 연다.
int shm_open(const char *name, int oflag, mode_t mode)
↳SM이름 ↳open_flag ↳접근
성공시 : return file descriptor(특정한 파일에 접근하기 위한 추상적인 키) - 음수가 아닌 수
실패시: return -1

Shared Mem 제거 (shm_open과 반대연산)
int shm_unlink(const char *name)
↳SM이름
성공시 : return 0
실패시 : return -1

=> 둘 다 #include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>

descriptor로 파일크기 변경 int ftruncate(int fildes, off_t length)
↳file descriptor ↳파일 크기
성공시 : return 0 (+) truncate : file명으로 크기 변경
실패시 : return -1
#include <unistd.h>

fd로 지정된 file or 객체에서 offset을 시작으로 length byte 만큼을 start 주소로 대응시키도록 한다.

void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)
↳kernel에게 file 어디에 mapping:보통0 ↳mapping 시킬 Mem 영역 길이 ↳mapping 유형, 동작 specify ↳mapping에 원하는 Mem 보호조건 ↳file descriptor ↳mapping시 len의 시작점

(+) prot
PROT_READ : 읽기 가능 page
PROT_WRITE : 쓰기 가능 page
PROT_EXEC : 실행 가능 page
PROT_NONE : 접근불가 page

성공시 : return 대응된 영역의 pointer
실패시 : return -1

mmap()으로 만들어진 mapping 제거
int munmap(void * start, size_t length)
start에서 시작하는 process 주소공간에 위치한 page를 포함하는 mapping 제거
이 영역은 length byte 만큼 이어져야 한다.
성공시 : return 0
실패시 : return -1

=> 둘 다 #include <sys/mman.h>
#include <unistd.h>

# Problem 2

- Write a program that display students' information by inter-process communication by POSIX shared memory.
  - Parent process writes students' information on a structure array on a shared memory
    - Parent does not display any text output.
  - Child process reads the students' information from the shared memory and display them

↳ 일단 parent가 shared Mem 넣고 child가 읽어와서 print.

```
$ gcc hw4_2.c -o hw4_2 -lrt      # be careful not to make a typo
$ ./hw4_2
name = Peter Parker, id = 2200001, major = Computer Science
name = Natasha Romanoff, id = 2200002, major = Electric Engineering
name = Clark Kent, id = 2200003, major = GLS
name = Diana Prince, id = 2200004, major = CSEE
```

# Problem 2

- Constants

```
#define FILENAME "shm_hw4_2_<your_id>"
    // e.g., "hw4_2_220001.shm"
    // after execution, display the contents of /dev/shm/<FILENAME>
    // $ od -x /dev/shm/hw4_2_220001.shm

#define SIZE 1024          // the size of Student array
```

- Data structure

```
typedef struct {
    char name[32];         // name
    char id[32];           // student id
    char major[32];        // major
} Student;
```

# Problem 2

- Parent process
  - Create a shared file FILENAME using shm_open()
  - Set the file size to SIZE * sizeof(Student) using ftruncate() → Student 구조체가 1024개까지 만들수 O
  - Create a child process
    - On failure, display an error message and quit.
  - Map the shared file to the memory space using PROT_WRITE
    - Take it using a Student pointer variable
      Ex) Student *s = mmap(…); parameter 찾아보기
  - Fill s[0], …, s[3] with the following information
    - s[0]: (name = Peter Parker, id = 2200001, major = Computer Science)
    - s[1]: (name = Natasha Romanoff, id = 2200002, major = Electric Engineering)
    - s[2]: (name = Clark Kent, id = 2200003, major = GLS)
    - s[3]: (name = Diana Prince, id = 2200004, major = CSEE)
      // Assume more students can be added in the future
    - To indicate the end of array, fill the id of s[$n$] with empty strings "", where $n$ is the number of students
  - // Do not display any message from parent process

- Child process
  - Map the shared file to the memory space using PROT_~~WRITE~~ READ · read 모드
    - Take it using a Student pointer variable (similar to the parent)
  - Display s[$i$]'s until s[$i$].id is an empty string ($i$ >= 0) → s[i].id가 빈 string이면 loop 탈출
    - s is the array of Students in the shared memory block
  - Unlink the shared file using unlink()

# Problem 3

- Implement a delayed key buffer using inter-process communication using POSIX shared memory
  - Producer
    - Reads keys from the user and put them into the shared memory buffer.
      - Use 'scanf(" %c", &key);' to ignore the Enter key.
      - Whenever a key is inserted, display the state of the shared buffer as the example
    - When the user types ESC,
      - Put '₩0' character into the buffer twice to lead the consumer to terminate after displaying all characters in the buffer.
      - Terminate
  - Consumer
    - Retrieves keys and display it only when there are three or more keys in the buffer.
      - Whenever a key is ~~inserted~~ retrieved, display the state of the shared buffer as the example
    - If the retrieved key is '₩0', terminate

- Compilation

  $ gcc hw4_3.c -o hw4_3 -lrt

  use shared Mem

# Problem 3

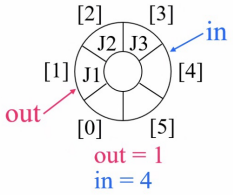
- Example (blue fonts indicate user's input)

```
$ ./hw4_3
Type keys. Press ESC to end.
a                                   // buffer에 insert
[producer] inserting key = a (IsEmpty = 0, IsFull = 0, KeyCount = 1, in = 1, out = 0)   // buffer 상태
b
[producer] inserting key = b (IsEmpty = 0, IsFull = 0, KeyCount = 2, in = 2, out = 0)
1
[producer] inserting key = 1 (IsEmpty = 0, IsFull = 0, KeyCount = 3, in = 3, out = 0)
==> [consumer] retrieved key = a (IsEmpty = 0, IsFull = 0, KeyCount = 2, in = 3, out = 1)
2
==> [consumer] retrieved key = b (IsEmpty = 0, IsFull = 0, KeyCount = 2, in = 4, out = 2)
[producer] inserting key = 2 (IsEmpty = 0, IsFull = 0, KeyCount = 3, in = 4, out = 1)
3
==> [consumer] retrieved key = 1 (IsEmpty = 0, IsFull = 0, KeyCount = 2, in = 5, out = 3)
[producer] inserting key = 3 (IsEmpty = 0, IsFull = 0, KeyCount = 3, in = 5, out = 2)
X
==> [consumer] retrieved key = 2 (IsEmpty = 0, IsFull = 0, KeyCount = 2, in = 6, out = 4)
[producer] inserting key = X (IsEmpty = 0, IsFull = 0, KeyCount = 3, in = 6, out = 3)
Y
==> [consumer] retrieved key = 3 (IsEmpty = 0, IsFull = 0, KeyCount = 2, in = 7, out = 5)
[producer] inserting key = Y (IsEmpty = 0, IsFull = 0, KeyCount = 3, in = 7, out = 4)
^[                                  // ESC
Terminating producer.
==> [consumer] retrieved key = X (IsEmpty = 0, IsFull = 0, KeyCount = 4, in = 10, out = 6)
==> [consumer] retrieved key = Y (IsEmpty = 0, IsFull = 0, KeyCount = 3, in = 10, out = 7)
Terminating consumer.
```

# Producer–Consumer Problem using Bounded Buffer

- Representation of buffer
  - Buffer is represented by circular queue

```
#define BUFFER_SIZE 6
typedef struct {
   . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;          // tail or rear
int out = 0;         // head or front
```



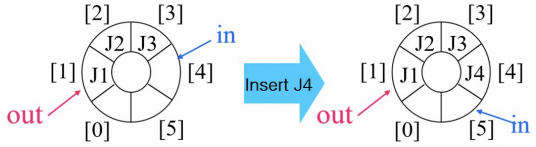

  - Empty/full condition
    - in == out: buffer is empty
    - (in+1)%BUFFER_SIZE == out: buffer is full

Cf. Buffer can store at most BUFFER_SIZE – 1 items

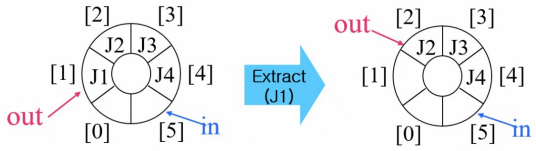
get key count

# Circular Queue

- Circular queue: fixed-size buffer whose logical structure is circular
  - Last element is followed by first element
- Inserting an item
  - buffer[in] = newItem;
  - in = (in + 1) % n;




- Extracting an item
  - item = buffer[out];
  - out = (out + 1) % n;



# Problem 3

- KeyBuffer

```
#define FILENAME "hw4_3_<your id>.shm“        // e.g., “hw4_3_220001.shm”

#define BUFFER_SIZE 128

#define TRUE 1
#define FALSE 0

typedef struct {
    char buffer[BUFFER_SIZE];
    int in, out;
} KeyBuffer;

// TO DO: implement the following functions
void InitBuffer(KeyBuffer *buf);          // set in and out to zero
int IsEmpty(KeyBuffer *buf);              // return 1 if buf is empty, otherwise, 0
int IsFull(KeyBuffer *buf);               // return 1 if buf is full, otherwise, 0
int GetKeyCount(KeyBuffer *buf);          // return the number of keys in the buffer
    ↳ in - out ; in < out 때는 예외처리!   // computed the number of keys from in and out
void InsertKey(KeyBuffer *buf, char key); // insert a key into buf
char DeleteKey(KeyBuffer *buf);           // delete a key from buf and return the deleted key
```

# Problem 3

- main()
  - Create a shared file FILENAME using shm_open().
  - Set the size of the shared file to sizeof(KeyBuffer) using ftruncate().
  - Create a child process
    - On failure, display an error message and terminate. → ERROR check
  - On parent process, call Producer()
  - On child process, call Consumer()
- Implement the following two functions
  - void Producer(int shm_fd);
  - void Consumer(int shm_fd);

# Problem 3

- void Producer(int shm_fd);
  - Map shm_fd to its memory space using mmap().
    - Use PROT_WRITE | PROT_READ.

    Ex) KeyBuffer *buf = mmap(…);
  - Initialize *buf* by InitBuffer().
  - Repeat
    - Read a key from the user.
    - If the key is ESC (27), insert '₩0' into *buf* twice. (If *buf* is full, wait.)
    - Otherwise, insert the key into *buf*
    - Display the key and the state of *buf*
- void Consumer(int shm_fd);
  - Map *shm_fd* to its memory space using mmap()
    - Use PROT_WRITE | PROT_READ.
  - Repeat
    - If *buf* contains less then three keys, wait.
    - Retrieve a key from *buf* using DeleteKey().
    - If the retrieved key is ESC (27), break the loop.
    - Otherwise, display the retrieved key and the state of *buf*.
  - Unlink the shared file